Freeform Search

Database: US Pre-Grant Publication Full-Text Database; US Patents Full-Text Database; US OCR Full-Text Database; EPO Abstracts Database; JPO Abstracts Database; Derwent World Patents Index; IBM Technical Disclosure Bulletins

Term:

Display: 10 **Documents in Display Format:** - **Starting with Number** 1

Generate: ○ **Hit List** ◉ **Hit Count** ○ **Side by Side** ○ **Image**

Search Clear Interrupt

Search History

DATE: Monday, May 17, 2004 Printable Copy Create Case

Set Name side by side	Query	Hit Count	Set Name result set
DB=PGPB,USPT,USOC,EPAB,JPAB,DWPI,TDBD; PLUR=YES; OP=OR			
L22	L18 and 707/200	158	L22
L21	L20 and 707/200	8	L21
L20	L19 and size	45	L20
L19	L18 and unused near (space or slot)	48	L19
L18	L17 and key	2556	L18
L17	L16 and (data with records or data near records)	3443	L17
L16	L15 and (sections or partitions)	7168	L16
L15	(database or data with base) near manag$	21727	L15
DB=USPT; PLUR=YES; OP=OR			
L14	5390359.pn.	1	L14
L13	5390359.pn.	1	L13
L12	5914938.pn.	1	L12
L11	5914938.pn.	1	L11
L10	5897637.pn.	1	L10
L9	5897637.pn.	1	L9
L8	5914938.pn.	1	L8

L7	5974421.pn.	1	L7

DB=PGPB,USPT,USOC,EPAB,JPAB,DWPI,TDBD; PLUR=YES; OP=OR

L6	5809494.uref.	12	L6
L5	5893120.uref.	5	L5
L4	6442553.uref.	0	L4
L3	5809494.pn.	2	L3
L2	5893120.pn.	2	L2
L1	6442553.pn.	2	L1

END OF SEARCH HISTORY

Refine Search

Search Results -

Terms	Documents
L21 and (sections or partitions or spaces)	6

Database:
US Pre-Grant Publication Full-Text Database
US Patents Full-Text Database
US OCR Full-Text Database
EPO Abstracts Database
JPO Abstracts Database
Derwent World Patents Index
IBM Technical Disclosure Bulletins

Search:

Refine Search

Recall Text | Clear | Interrupt

Search History

DATE: Monday, May 17, 2004 Printable Copy Create Case

Set Name side by side	Query	Hit Count	Set Name result set
DB=PGPB,USPT,USOC,EPAB,JPAB,DWPI,TDBD; PLUR=YES; OP=OR			
L23	L21 and (sections or partitions or spaces)	6	L23
L22	L21 and (sections or partitions)	5	L22
L21	L20 and data near record	7	L21
L20	(trash near manag$ or garbage near manag$)	126	L20
L19	L18 and (trash near manag$ or garbage near manag$)	11	L19
L18	(database or data with base) near manag$	21727	L18
L17	L16 and database near manag$	85	L17
L16	garbage near3 manag$	448	L16
L15	711/160	324	L15
L14	711/130	624	L14
L13	711/129	708	L13
L12	711/117	863	L12
L11	711/113	1497	L11
L10	711.clas.	21250	L10

L9	707.clas.	20403	L9
L8	707/103	3101	L8
L7	707/8	1939	L7
L6	707/3	6309	L6
L5	707/1	6381	L5
L4	707/200	3251	L4
L3	707/204	1771	L3
L2	707/205	1589	L2
L1	707/206	892	L1

END OF SEARCH HISTORY

L22: Entry 133 of 158 File: USPT Mar 9, 1999

DOCUMENT-IDENTIFIER: US 5881379 A
** See image for Certificate of Correction **
TITLE: System, method, and program for using duplicated direct pointer sets in keyed database records to enhance data recoverability without logging

Abstract Text (1):
The system, method, and program product of this invention allows a database management system to internally use direct and indirect pointing to locate targeted data elements that are logically related to another data element or are a target of a secondary index. By using direct and indirect pointing, the number of steps involved in a reorganization of the database can be reduced. After a reorganization, the database management system does not go back, in a separate process, to update all of the direct pointers that have pointed to segments that have moved as a result of the reorganization. Instead, the direct pointer is updated, by using the indirect pointer, only upon a first reference to the targeted data element that has moved. Each targeted data element has an entry in an indirect index where a unique identifier of the targeted data element is the key into the index. The indirect index has two direct pointer slots. Which pointer slot is used by the DBMS is determined by the odd/even reorganization number. For each reorganization, the DBMS alternates between the two portions of the indirect index to update the appropriate one of the two pointers with a new location whenever the targeted data element is moved. As a result, the indirect index contains the most recent update and the most recent previous update. If the reorganization fails, the most recent previous pointer is used to find the location of the targeted data element. Consequently, as the reorganization process is proceeding, the update to the assigned slot for the new reorganization number can be made without logging and without involving unnecessary overhead associated with maintaining direct recoverability of the keyed data records.

Brief Summary Text (6):
This invention relates to database management systems, methods, and programs, and more specifically to the reorganization of hierarchical data involving data that is logically related and data that is the target of indexes, and even more specifically, to the recoverability without logging of information resulting from a failed reorganization.

Brief Summary Text (10):
A database can become disorganized to the point where a different organization of the data elements may result in more efficient operations on the data, more efficient use of data storage, and increased data capacity. The solution then is to perform a reorganization on the data. Reorganization of a database includes changing some aspect of the logical and/or physical arrangement of the database. Any database management system (DBMS) will require some type of reorganization in order to restore a given level of performance and to improve the degraded capacity of the database. One type of reorganization involves the restoration of clustering and removal of overflows as described above.

Brief Summary Text (11):
Another type of reorganization involves splitting up a full partition of data. Since a 32 bit relative byte address is limited to 4 gigabytes, when a database

exceeds 4 gigabytes of data, it has to be partitioned into two or more physical data sets. When a partition is full, it has to undergo a reorganization to get more space.

Brief Summary Text (13):
A hierarchical database management system, such as IBM's IMS DL/I, manages data in a tree structure. Each data element is called a segment, and the first data element is called the root segment of the structure. The root segment is the top of the tree and each subordinate segment is a child of the root or is a child of the child of the root segment or is a child somewhere further down the lineage, i.e., tree. One database is analogous to a forest containing a lot of trees all of which share the same defined segment structure.

Brief Summary Text (14):
IMS DL/I also has keyed access directly to any segment in the database. Every segment could be pointed to by an index. An index is similar to an index in a book that provides readers with one relatively quick means of locating a section of interest. An index is a listing that can derive the location of a desired element, eliminating the need for a "brute-force" sequential search through the collection of elements, thereby providing an alternative form of data access. For example, a root segment could be a serial number, part number, an account number, etc., and the first dependent segment of the root might be a name. An index on the name provides direct access to specific data without first knowing the identification of the root segment or segments that have dependent segments with a given name.

Brief Summary Text (15):
Indexing data by using direct pointers to, and between, data elements is common in databases managed by Database Management Systems (DBMS). It is well known that one can point indirectly to something through an index that can be pointed to directly. However, indexing data is usually the result of defining, by the users of such systems, either beforehand or dynamically, the data for which an index is used.

Brief Summary Text (20):
Presently, it takes a significant amount of time, and multiple steps, to reorganize a database that consists of direct pointers to other data elements in the database since the direct pointers have to be updated after a reorganization. The database management system does not know in advance which data element the pointer is going to point to, so the database management system has to go back later and update the pointers after the reorganization. Reorganization of such a database is a multi step process and is very time consuming.

Brief Summary Text (29):
One method for finding data moved by a reorganization process uses the fully concatenated key of the target of a logical relationship or secondary index. However, this method requires unique keyed data. A method is needed that allows non-unique and un-keyed data to exist in the database.

Brief Summary Text (30):
It is also desirable to reduce the contention between other parts of the system that are not being impacted by the reorganization directly, such as a secondary index. In general, when a database is being reorganized that has alternate, i.e., secondary, indexes associated with a data element being moved, at the time the data element is being moved from the old location to the new location, there is an ability to update all of the indexes. This is because they are known at that time, and the index that is indexing into that point is based on the data itself that is being moved, i.e., contained in effect within the database record. In general, the secondary index is based on data values in that data element. In the IMS DL/I database, the database element can be indexed either on a data value within that element or any element in its dependency tree. During reload, all of the information needed is available to directly update each secondary index at the time

of reload. However, updating a secondary index at this time creates recoverability difficulties if a reorganization fails. These recoverability difficulties are reduced if the index is updated after the relocated data elements have been successfully completed.

Brief Summary Text (33):
It is an object of this invention to allow non-disruptive reorganization of user data managed by a database management system.

Brief Summary Text (41):
As such, the preferred embodiment of this invention assigns a unique identifier (id) to every segment, i.e., data element, in the database at the time the segment is created, i.e., either at initial load when the database is first created or when the segment is added to the database through an update. The id is maintained for life across multiple reorganizations, i.e., the unique id does not change during a reorganization. Through the database definition for any given segment, it is known whether a segment is independent, i.e., whether it ever points outside of the database record or an outside database record points to it, i.e., whether it is a logically related segment, or whether it is a target of a secondary index. For such segments that can be pointed to, the unique id is used as a key of an index data set, i.e., a locator file.

Brief Summary Text (42):
Each such segment has a locator file, i.e., an indirect list entry (ILE), associated with it wherein the key to the locator file is the unique id of the segment. The locator file contains the new location and old location, i.e., the relative byte address (RBA), of the segment and its physical parent. By examining the database definition for any given segment, it can be determined whether the segment is involved in a logical relationship or is the target of a secondary index, and, therefore, whether the locator file for that segment should be maintained if the segment is moved.

Brief Summary Text (43):
Also, for every partition of the database, the database management system keeps a table in memory of the current reorganization number associated with each partition. Whenever a partition is reorganized, the reorganization number increases.

Brief Summary Text (44):
If a given segment points to a target segment, the given segment will have a pointer set associated with it having the pointer itself, a description of the partition of the database where the target exists to give a context, and a reorganization number to indicate the reorganization level of the target at the time the direct pointer is assigned. More specifically, the fields of the pointer set include a relative byte address of the target, the unique id of the target and the reorganization number. At actual execution time, the table in memory is used to verify whether a reorganization number in a pointer set matches the current reorganization number in the instorage table to indicate whether the direct pointer was valid and could be used. If the reorganization number matches the reorganization number in memory for that targeted partition, the RBA for the target in the pointer set is valid and can be used, i.e., the direct pointer in the pointer set can be used for direct reference. If the reorganization numbers do not match, this means that the target partition has been reorganized since the direct pointer has been last used so the direct pointer is out of date. Therefore, the RBA for the target in the pointer set is not valid and can not be used.

Brief Summary Text (45):
If the RBA for the target is not valid, i.e., the reorg number in the pointer set does not match the reorg number in memory for that target partition, then the current location of the target data element is resolved as follows. The unique id

of the target segment in the pointer set will be used as a key into a locator file for that target segment. The locator file will maintain a current RBA location for the target segment which can be used to update the RBA location for the target in the pointer set. Once updated, and from this point on, until another reorganization which moves this segment, the pointer set referencing that segment will be correct and the direct pointer can be used for direct reference. The direct pointer is corrected only if it is used so machine time is not wasted if it is not in use, but yet the current location can always be found.

Brief Summary Text (46):
This invention is applicable not only to logically related hierarchical data, but also to data that is the target of a secondary index. In a partitioned database, i.e., a database that is broken up into a series of logically related records that can be operated on independently, a single partition can be reorganized while the rest of the database is available for use. Since the secondary index is organized on the value that is being indexed, the secondary index spans the entire database so that trying to update the index directly does not isolate changes to that part of the data set associated with the partition but would have impact on, data relating to other partitions. For performance reasons and for reducing contention, the secondary index also uses a pointer set to provide the capability to point directly, or to use the locator file to find a current location and to correct the index on reference. This allows the reorganization to be non-disruptive, i.e., not to affect the rest of the system.

Brief Summary Text (47):
As such, this invention allows non-disruptive reorganization of data by using a method of dual pointing with direct and indirect pointing to logically related data and data that is a target of secondary indexes. Direct and indirect pointing allows data element(s) to be relocated and stored in new physical location(s) without the need to immediately find and update the data elements that point to each moved data element. More specifically, this invention uses indexing within a Database Management System to improve data availability.

Drawing Description Text (7):
FIG. 2 shows the structure of an indirect list entry key;

Detailed Description Text (2):
FIG. 5 shows the components of a database system including data, hardware, software, and users. Although FIG. 5 illustrates a single system, various components, e.g., the applications, end users, storage devices, processors, memory, etc., or multiple ones of any of the components could be interconnected in a network through a communications link. The system 500 allows multiple users 591 to access the database 510, 511 concurrently. A user can be an end user 591 who interacts with the system from an on-line workstation or terminal through an on-line application 582 or an interface provided by the DBMS 541, or an user can be a conventional batch application 581 sending requests to the DBMS. The data can reside in one database or multiple databases 510, 511. The hardware portions of the system consist of secondary storage volumes 510, 511, such as moving head magnetic disks, that are used to hold the stored data, together with the associated I/O devices, device controllers, I/O channels, etc., and processors 521 and associated main memory 531 that are used to support the execution of the database system software 541. Data structures and programs used by the database system software may reside in any of the many forms of memory including main memory, RAM, ROM, and physical external storage volumes. Between the physical database itself (i.e., the data as actually stored) and the users 591, 581, of the system is a layer of software, the database management system (DBMS) that runs on top of the operating system 571. All requests from users for access to the database are handled by the DBMS. The DBMS shields the database user from hardware-level details. The DBMS also contains software components that interact with the hardware components of the system to manage reorganizations and to manage the use of pointers as described

hereinbelow.

Detailed Description Text (4):
This invention involves a system, method and program product of dual pointing called direct and indirect pointing by using a pointer set that has a direct pointer that can be used when valid or by using an indirect pointer to a locator file when the direct pointer is not valid. This invention allows non-disruptive (i.e., other parts of the system remain unaffected) reorganization, of user data managed by a database management system by using internal indexing. There are benefits of a database management system using indexing internally by providing data availability and performance.

Detailed Description Text (8):
The reorganization cursor information must be on some logical data boundaries as opposed to physical relative byte addresses (RBA's). The logical boundary may take advantage of the data organization (e.g., one or more HDAM root anchor points or one or more root keys in a HIDAM database) but is not restricted to such boundaries.

Detailed Description Text (12):
Therefore, in a preferred embodiment, indirect access to data that has been physically relocated is then done by using a very large table entry number as the key of a record in an index. This is an internally managed secondary index that provides a unique method of indirect pointing allowing data availability before, during, and after a non-disruptive reorganization process on the data.

Detailed Description Text (13):
More specifically, the invention is described with reference to FIGS. 1A, 1B, 1C, and 1D. DBY 110 and DBZ 160 are two different databases both having separate definitions. However, their definitions may allow a logical relationship to exist. For example, in databases where there is a concept of pairing, child 111 in database DBY 110 and Child 161 in database DBZ 160 are related. The logical parent 162 of child 111 may be the physical parent 162 of child 161. Likewise, the logical parent 112 of child 161 may be the physical parent 112 of child 111. An example of logical relationships is in a manufacturing bill of materials structure where the data elements 111 in database DBY 110 represent parts that make up a subassembly 112. The different instances of data elements 111, 112 may represent engineering change levels. The other database, DBZ 160, may be the vendors that supply the parts. The customer of the database defines the meaning and the semantics of the logical relationship. The database management system supplies the pointers, in accordance with the customer definitions, to make the database run fast.

Detailed Description Text (14):
In implementing direct pointing, for each segment, i.e., data element, there is a data portion 139 that the user sees, and a prefix 131 that consists of a series of pointers. One of the items is a logical pair pointer 133. By looking at the definition of the database, it can be determined where to go in the prefix 131 to get the direct pointer. The prefix is allocated in four (4) byte blocks, where each of the four byte blocks represents a relative byte address (RBA). The database definition describes what each block is. The pointer set 140 is a series of these blocks. If there is partitioning, the logical pair pointer 133 is expanded to indicate the partition id 144 as well as the relative byte address (RBA) 142 to indicate the data set in finding the logical pointer 133. The partition id 144 defines which data set the logical pointer is in, and the RBA 142 is the location within the data set. A data set is treated as if it is a continuous byte stream where every byte is not named, but if the physical location of the partition is known, then the byte location is known.

Detailed Description Text (15):
For the example as shown in FIGS. 1A, 1B, 1C, and 1D, it is assumed that data

element 111 and data element 161 are related in such a way that the data portions are identical. The following describes features of the invention that overcomes the difficulties in managing these two logically related data elements. A pointer set 140 for data element 111 will have in it the partition id 144 of the target 161, the RBA 141 of the logical parent 162, the RBA 142 of the twin 161, the unique id 170 for the twin data element 161 and the reorg number 150 for that partition containing the data element 161. With the direct pointer set 140 for data element 111, a database management system can go directly from data element 111 to the twin data element 161 or the parent data element 162. The database management system will also have the unique id 170 of the twin 161 and a reorg number for that twin which is set to zero (0) initially.

Detailed Description Text (16):
Upon a reorganization to the partition in DBZ 160, there is a separate locator file 180, also referred to as an indirect list entry (see FIG. 3), where the unique id 170 of twin 161 is the key 187 to the locator file 180. In the locator file 180 for data segment 161, there is the RBA 181 of the physical parent 162 and the RBA 182 of the data segment 161. The locator file 180 is a special form of a symbolic because it does not go through a root via a concatenated key. It is a symbolic that goes directly into something. However, in this preferred embodiment, it is only usable for this process, it is not a general symbolic for other external programming use.

Detailed Description Text (17):
During operation, if the database management system (DBMS) reaches data element 111, as a result of the way a user formulated a call, etc., and wants to access the twin data segment 161, the DBMS looks at the partition id 144 for the twin data element 161 in the pointer set 140. The DBMS uses this partition id to look up in a table in memory a reorg number that is associated with that partition id. Each partition has its own reorg number. As each partition is reorganized, the reorg number associated with that partition is updated by incrementing it in the table in memory. When the database is opened, the reorg numbers are put into the table in memory. Whatever partition is being used, the current reorg number is available to the DBMS. The DBMS compares the reorg number in memory for that partition with the reorg number 150 in the pointer set 140.

Detailed Description Text (18):
If the reorg number 150 matches what is in memory, the DBMS can use the direct pointer 142 in the pointer set 140 to get to the twin data element 161. If the reorg number 150 in the pointer set 140 is incorrect or does not match, then the fields in the pointer set 140 have to be updated. The DBMS uses the unique id 170 in the pointer set 140 as a key 187 into the locator file 180 to get the RBA 181 of the physical parent 162 to update the pointer set 140. The DBMS puts the new RBA 181 from the locator file 180 as the RBA 141 of the logical parent in the pointer set 140. The DBMS also puts the RBA 182 of the twin data element's 161 own location as indicated in locator file 180 into the RBA 142 of the twin location in pointer set 140. The reorg number 150 is also updated.

Detailed Description Text (20):
In a partitioned database, i.e., a database that is broken up into a series of logically related records that can be operated on independently, a single partition can be reorganized while the rest of the database is available for use. Since the secondary index is organized on the value being indexed, the secondary index spans the entire database. Trying to update the index directly does not isolate changes to that part of the data set associated with the partition but would have impact on data relating to other partitions. For performance reasons and for reducing contention, the secondary index also uses a pointer set to provide the capability to point directly to, or to use the locator file to find a current location. This reduces contention and corrects the index as it is being referenced which allows the reorganization to be non-disruptive, i.e., not to affect the rest of the

system.

Detailed Description Text (21):
The locator file, as described above, is structured and formatted in a similar way as an index is structured and formatted. For clarification, an index database refers to one or more elements containing a key for the purpose of finding a pointer to some user data, and these keyed elements are managed in key sequence such as in an IBM VSAM KSDS. The same DBMS internal structure that manages secondary indexes is used to manage the locator file. Structuring the locator file in this way saves development cost, allows code reuse, and provides for other efficiencies, such as in managing it.

Detailed Description Text (22):
Indirect List Entry Keys (ILKs) and The Indirect List Entry

Detailed Description Text (23):
An unique id 170 FIG. 1A, also referred to as an indirect list entry key (ILK) 170 FIG. 2, is created for each segment at the time each segment is created. There are at least two ways of doing this for each segment that is placed in the database. By the database definition for that segment it can be determined whether the segment is independent, i.e., it does not participate in a logical relationship by pointing to another segment or being pointed to by another segment or is not the subject of a secondary index. If it is not independent, i.e., it is dependent, it will require a locator file and a unique id should be created for it. If it is independent, then a unique id does not need to be created for it. However, this approach may require additional process steps. For example, at the time a database is defined, there may be three secondary indexes. After running for a while, it may be discovered that some applications may be able to improve their efficiency by the addition of a fourth secondary index. To do this, the database would have to be redefined to add the fourth secondary index by unloading (in effect a reorganization), reloading, and assigning the unique ids to existing segments that have now become a target.

Detailed Description Text (26):
The general format of the unique id (ILK) 170 discussed above is shown in FIG. 2. When a partitioned segment, such as a DL/I segment, is inserted, the ILK 170 is formed from the relative byte address (RBA) 171 of that segment, the current partition id 172, and the current partition reorg number 173. Partitioned DL/I relies on the use of an indirect list entry key (an ILK) 170 for various purposes as further described.

Detailed Description Text (27):
FIG. 3 illustrates the structure of the pointer set 140 and the indirect list entry, i.e., locator file 180. Indirect list entries (ILEs) 180 and pointer sets 140 are used to support logical relationships, which can span partitions, and secondary indexes which can also span partitions. The pointer set is an expanded pointer containing pointer information for both directly and indirectly locating a segment. The indirect list entry is an entity pointed to indirectly that always contains the pointer set information for locating a segment via direct relative byte addresses (RBA). Reviewing briefly the function of these elements, any segment which has the ability to point outside its own database record uses a pointer set 140, and the "pointed to" segment must have an entry in the ILE data set as the pointing mechanism. Consistent with this, any segment which is pointed to from outside the database record has an associated ILE 180, i.e., locator file, and contains the key of the ILE (ILK) 187 so that the ILE 180 may be updated when the segment is moved.

Detailed Description Text (28):
By examining the database definition for any given segment, it can be determined whether a segment is independent or not, i.e., whether or not it ever points outside of the database record or whether or not an outside database record points

into it, i.e., whether or not it is a logically related segment, or whether it is a target of a secondary index. Therefore, by examining the database definition for that segment, it can be determined whether or not an ILE 180, i.e., a locator file, needs to be maintained for that segment and whether or not it will need to be updated if the segment is moved. When a pointed-to segment is initially inserted, the ILK 170 of that segment is placed in the pointer set 140 of the segment pointing to it. The same RBA 142/partition ID 144/reorg number 150 is also stored as the current direct pointer information to the pointed to segment. If the segment is not the target of a secondary index, or does not participate in a logical relationship, then the ILE 180 for that segment does not have to be affected. The reorganization is independent of it, though the segment still keeps its unique id. When the partition containing the pointed to segment is reorganized then the ILE (whose key is the original ILK) 180 is updated with the new and current location and other associated information to point to the moved target segment. In a later operation using a logical relationship or secondary index that points to the segment through a pointer set 140, it can be determined, by checking the reorg number 150, whether its pointer 140 is out of date, and can go to the locator file 180 to correct it.

Detailed Description Text (31):
FIG. 6 illustrates a preferred embodiment of the structure of the indirect list entry 180 which illustrates another aspect of this invention for handling recoverability. When a reorganization is started on a partition, the indirect list entry 180 is updated with the new location 182. However, a reorganization is an atomic operation, i.e., if it fails to complete it is as if it never happened. Nevertheless, the indirect list entry 180 would have been updated. Consequently, the indirect list entry 180 will not be correct. It will not reflect the real location of the data at this point. It is undesirable to have to log all of the changes to the locator file in order to back out the changes in the event of a reorganization failure.

Detailed Description Text (32):
The specific structure of the locator file in a computer usable medium accessible by the database manager allows the data set for the new location of the segment to be discarded, and updates to the ILE started over with, without any additional system overhead if the reorganization process fails.

Detailed Description Text (33):
Duplicate direct pointers are maintained in the indirect keyed data record, i.e., the ILE. The keyed data record itself has two direct pointer slots. Which pointer slot is used is determined by the odd/even reorganization number. As the reorganization process proceeds, the update to the assigned slot for the new reorganization number can be made without logging and without involving unnecessary overhead associated with maintaining direct recoverability of the keyed data records. This assumes the restart and recovery of data reorganization is managed on some logical/physical boundaries of the targeted data. A further assumption involves the physical recovery of the particular keyed data. That is, the keyed data set is rebuilt from a pass through of the targeted data instead of employing change logging and recovering from logged changes.

Detailed Description Text (35):
The reorganization number for each partition stored in a table in memory, which is also the same reorganization number 150 stored in the new location half of the indirect list entry, is an external monatomically increasing (increasing in one unit increments such that each sequential number alters between being odd or even) number. This provides the database management system the ability to determine which even or odd half of the indirect list entry to use as the current targeted segment location. If the current reorganization number is odd, but this reorganization has failed, the database management system goes to the location stored in the even half. For a subsequent reorganization, the database management system will store

the new location in the odd half. For a following reorganization, the database management system will flip over to the even half and store the new values there. The database management system flip flops between halves with each reorganization, i.e., as the external number monatomically increases. As such, old (i.e., most recent previous) values and new (i.e., newly changed) values are combined into a single file having separate portions, i.e., halves, for each, and is managed on a flip flop basis as an external number increases.

Detailed Description Text (37):
In the preferred embodiment, there is no need to perform any logging, locking, or backing out of any changes to the indirect list entry. There is also no need to allocate an additional temporary indirect list entry to record changes. Typically, for users currently opened against the partition, they would not only have to reallocate the data set to the reorganized data set, but they would also have to reallocate the indirect list entry. In the preferred embodiment of this invention, all of the system management overhead that is used for those steps is saved. If the reorganization fails, the database management system utilizes that part of the indirect list entry containing old values which have not been altered. The invention has maximum data integrity with minimum system management overhead and minimum execution overhead. If the indirect list entry data set is lost, it can be recreated by scanning the database on the fly.

Detailed Description Text (39):
FIGS. 4A and 4B shows the overall method steps, including a high level description of the program steps, of using direct and indirect pointers by a database management system for logically related data and data that is a target of a secondary index.

Detailed Description Text (40):
Step 400 indicates that the DBMS performs its usual processing as events or requests are received. Referring to FIG. 4A, at step 402, whenever a segment is added to the database, at initial database creation or by a later update to the database, that segment will be assigned a unique id. When a database is initially created, nothing, by definition points into it. An indirect list entry is created for a segment when it is added to a database, initially or during an update, if the segment is defined as a target of a index or participates in a logical relationship. The initial file would be considered at reorg number zero, i.e., just created and never reorganized. Therefore, the reorg number for the partition and for all pointer sets would be zero. The next event involves a reorganization of a partition. A reorganization is a logical read of the existing data so that one can recluster and eliminate overflow to get all of the data back close together.

Detailed Description Text (41):
The standard steps 404 in a reorganization include: reading the data in its logical order, i.e., an order having a desired cluster, recreating the data set with the maximum clustering for performance reasons, reestablishing distributed free space for performance reasons, reestablishing distributed free space for growth, and establishing overflow areas so the database can grow. Some databases that have a requirement of partitions, such as IMS DL/I, have additional reorganization responsibilities to take partitions that are full and to break them into two partitions to gain more space.

Detailed Description Text (42):
In addition to these standard reorganization techniques, the following steps 406, 408, 410, 412, 414, occur. When the data segment is placed in the new data set it will have a new relative byte address. When reorganized, at the time a segment is put in its new location, if the segment is the target of a secondary index or participates in a logical relationship, a unique identity is used as a key into the indirect list entry so that all of the information as to where the segment is now can be added to the indirect list entry.

Detailed Description Text (43):
Referring to FIG. 4B, the DBMS continues its processing, step 400. If a user request comes in that requires a target element to be accessed, step 420, the DBMS will look into the pointer set of the segment that points to the target to find the partition id of the targeted segment, step 422. The DBMS will use this partition id to look up into memory the reorg number associated with that partition id, and will compare the reorg number in memory with the reorg number in the pointer set. If the reorg numbers match, step 424, then the pointer set is valid and the DBMS can use the RBA in the pointer set to find the target segment directly, step 426. If a reorganization has taken place, all data elements, whether in the same partition, or any other partition or any other database that points to this moved targeted segment, has the old reorg number in their pointer set. An attempt to follow the direct pointer will fail the reorg number check, steps 422, 424. The pointer set would show a reorg number set at zero, but, in this example, this reorganization has the value of one. The unique id is used to go to the indirect list entry, step 430 to pull out the current information, and to update the information in the pointer set, step 432. In a preferred embodiment, the DBMS determines whether the reorg number in memory is odd or even, step 428, and then gets the target location from the corresponding odd or even half of the indirect list entry to update the pointer, step 432. The target element is then located using the updated pointer, step 434. In other words, when an instance of targeted data has been moved by a reorganization process, the indirect pointer is used to find a keyed data record which contains an updated direct pointer. All future uses, prior to another reorganization on this partition, would be direct without having to go to this indirect list entry. The overhead of correcting the pointer is paid only once when it is used. The whole database does not have to be scanned. Once the correct value is determined, it is saved for all future use.

Current US Cross Reference Classification (2):
707/200

Issued US Cross Reference Classification (1):
707/200

Field of Search Class/SubClass (1):
707/200

CLAIMS:

1. A database system for reorganizing a database having at least one related data element related to a targeted data element, the database system comprising:

means for relocating to a new physical storage location the targeted data element independently of updating a direct pointer associated with the related data element to indicate a location of the targeted data element;

an indirect index having a unique indirect list entry key associated with two direct pointers; the first pointer containing information relating to a relocation to the new physical storage location as a result of a most recent reorganization; and the second pointer for containing previous information relating to a previous relocation to a previous storage location resulting from a previous reorganization;

means for updating the indirect index with the new physical storage location of the targeted data element when the targeted data element is relocated; and

means for updating, using the updated indirect index, the direct pointer at a first reference of the targeted data element after the targeted data element is relocated.

3. A database system, having at least one data element, comprising:

means for maintaining a reorganization number representing an update state of the data element wherein each subsequent update state is alternatingly associated with an odd or even reorganization number;

a data record having a keyed entry for at least each data element that is dependent, each keyed entry having duplicate direct pointers, a first direct pointer associated with the odd reorganization number and a second direct pointer associated with the even reorganization number, for pointing to a location of the data element;

means for determining whether to update the first direct pointer or the second direct pointer during a current reorganization depending upon whether the current reorganization number is odd or even;

updating the entry of the data record independently of logging the update in a separate record, whereby the entry contains, after an update, a current location and a most recent previous location;

accessing the first direct pointer for a current location of the data element if the current reorganization number is even and a current reorganization fails; and

accessing the second direct pointer for the current location of the data element if the current reorganization number is odd and the current reorganization fails.

7. A method of reorganizing a database having at least one related data element related to a targeted data element, the method comprising:

relocating to a new physical storage location the targeted data element independently of updating a direct pointer associated with the related data element to indicate a location of the targeted data element;

updating, in an alternating fashion, an indirect index with the new physical storage location of the targeted data element when the targeted data element is relocated, the indirect index having a unique indirect list entry key associated with two direct pointers; the first pointer containing information relating to a relocation to the new physical storage location as a result of a most recent reorganization; and the second pointer for containing previous information relating to a previous relocation to a previous storage location resulting from a previous reorganization; and

updating, using the updated indirect index, the direct pointer at a first reference of the targeted data element after the targeted data element is relocated.

10. A memory, for use with a database management system, for accessing a targeted data element from a related data segment, the memory comprising:

a direct pointer data structure comprising:

a first reorganization number indicating a reorganization level of a partition containing the targeted data element;

a relative byte address of a first location of the targeted data element associated with the first reorganization number;

a unique segment identifier of the targeted data element;

an indirect index data structure having the unique segment identifier as a key into

the indirect index data structure comprising:

duplicated fields wherein a first set of fields is associated with an even reorganization number and a second set of fields is associated with an odd reorganization number, the first and second fields each containing:

the even or odd reorganization number indicating the reorganization level of the partition containing the targeted data element; and

a relative byte address of a second location of the targeted data element associated with the reorganization level indicated by the even or odd reorganization number;

whereby the indirect index has a most recent location of the targeted data element and a most recent previous location of the targeted data element.

11. A program product, for use with a database system having at least one related data element related to a targeted data element, the program product having program code, on a computer usable medium, comprising:

means for causing a determination that the targeted data element needs to be accessed from the related data element;

means for causing a use of a direct pointer data structure, associated with the related data element, having a first location of the targeted data element, a partition identifier of the targeted data element, and a first reorganization number associated with the location of the targeted data element for comparing the first reorganization number to a second reorganization number in memory for the targeted data element;

means for causing a use of the first location to locate the targeted data element if the first reorganization number and the second reorganization number are the same; and

means for causing a use of an indirect index data structure, by using the partition identifier as a key into the indirect data structure, the indirect index data structure having an odd and even portion reflecting the odd or even nature of the second reorganization number, the indirect index being updated with a newer location of the targeted data element in a portion associated with the odd or even nature of a latest reorganization number when the targeted data element is moved, to locate the targeted data element by using the newer location in the portion of the indirect index associated with the odd or even nature of the second reorganization number, if the first and second reorganization number are different.

L22: Entry 152 of 158 File: USPT Jan 14, 1997

DOCUMENT-IDENTIFIER: US 5594881 A
TITLE: System for updating modified pages of data object represented in concatenated multiple virtual address spaces

Abstract Text (1):
Method and means are provided for simulating a contiguous data space within a computer memory, and for placing and accessing data objects of various sizes within the simulated contiguous data space. Multiple, sub-data spaces are concatenated in such a way that each page and each sub-data space in the contiguous data space are uniquely identified. Data objects are placed in the contiguous data space and at the first reference to a page of the data object, only the segment containing the referenced page in the contiguous data space is mapped to the database storage disk. Once a data space page is mapped, the operating system can read the page into memory without requesting a disk operation from the database manager. On modifying a page, if the database disk page location is changed, the contiguous data space page is remapped without changing the page address in the data space. Also, modified data pages are rewritten to the database storage disk in an ongoing manner set by the user, instead of at intervals set by the operating system.

Brief Summary Text (3):
Data on storage disks exists in sets of fixed length records accessible in pages. The size of a page varies depending on the system in use. On some systems, a page is 4096 (4K) bytes. Also, on some computers, e.g., virtual machines, the operating system keeps track of disk space in blocks of 256 pages called segments. This is necessary because the hardware requires that a control block be maintained for each segment for virtual address translation. When data is transferred from auxiliary storage to the CPU, pages of data are transferred to a storage buffer in segments.

Brief Summary Text (7):
A typical database storage system comprises a directory disk, one or more data disks, and one or two log disks similar to the data disks. The directory disk contains information on the mapping of the database pages from virtual storage to their real physical location and other information describing the physical configuration of the data base. The data disks store data, while the log disks record transactions against the database.

Brief Summary Text (8):
In a database system, users are assigned logical pages on data disks to store data objects. A data object is a logical set of pages in the database. For example, in a relational database system, a data object may be viewed as a set of pages containing records of rows and columns of a table where each row is a separate record, and each column is a different data field. When a data object is created, entries are inserted in the database directory disk to indicate which data object pages contain data, and their physical location on a data disk. Initially, only directory space is taken, but as a user inserts data into a data object, pages are allocated on a data disk and the directory disk is updated to identify those pages.

Brief Summary Text (15):
U.S. Pat. No. 4,843,541 ("Logical Resource Partitioning of a Data Processing

System") discloses a method and means for partitioning the resources in a data processing system into a plurality of logical partitions. The main storage, expanded storage, the channel and sub-channel resources of the system are assigned to the different logical partitions in the system to enable a plurality of preferred guest programming systems to run simultaneously in the different partitions.

Drawing Description Text (2):
FIG. 1 shows key information maintained in a database directory relevant to the present invention;

Drawing Description Text (3):
FIG. 2 shows an overall representation of the key elements in this invention;

Detailed Description Text (2):
The invention is implemented using the database storage system shown in FIG. 1. It comprises a directory disk 1, one or more data disks 2, and one or two log disks 2.1 similar to the data disks 2. The database storage system is part of a computer system having a central processing unit (CPU) 2.2 and memory 2.3. The CPU 2.2 runs a database management system (DBMS) software program which manages the data stored in the storage devices.

Detailed Description Text (3):
The directory disk 1 contains information on the mapping of the database pages from virtual storage to their real physical locations and other information describing the physical configuration of the database. The data disks 2 store data, while the log disks 2.1 record transactions against the database.

Detailed Description Text (4):
In a database system, users are assigned logical pages on a data disk 2 to store data objects. A data object is a logical set of pages in the database. For example, in a relational database system, a data object may be viewed as a set of pages containing records of rows and columns of a table where each row is a separate record, and each column is a different data field. As shown in FIG. 1, when a data object is created, entries are inserted in the directory disk 1 to indicate which data object pages contain data 3, and their physical location on a data disk 4.

Detailed Description Text (6):
The invention is summarized as comprising the following steps which are described in more detail in subsequent sections:

Detailed Description Text (11):
Details on the above steps are provide in the following sections.

Detailed Description Text (34):
Step 3. If the sub-data space 608a-608q was not created as determined in step 1006, then proceed to step 1008 which is further described in FIG. 12. Proceeding to step 1204, the database manager creates the sub-data space 608a-608q as previously described and sets the data space identifier value used to address that sub-data space 608a-608q. Continuing to step 1206, the database manager allocates and initializes the segment.sub.-- valid bit map 902. Processing continues to steps 1208 and 1210 which are described in detail below. Processing then returns from step 1008 and continues to step 1010; and

Detailed Description Text (38):
Checking the "segment valid" bit at each page request is inexpensive because the segment.sub.-- valid bit map 902 is small. If the segment 616a-616p is not mapped, then the database manager program reads the directory 1 blocks containing information on the pages required for that segment 616a-616p and issues a request to map the segment 616a-616p data space pages that contain the referenced pages to

page locations on disk 2. Once mapped, the "segment valid bit" is turned "ON" and any subsequent reference to read a page in that segment 616a-616p is translated directly to a virtual storage address without reference to the database directory 1. The net result is an improved query response time.

Detailed Description Text (39):
If a request is made for an update (the page was modified), the page may have to be shadowed (moved to a new disk 2 location). Alternatively, it may have to be inserted if it had not existed before (i.e. first insert). This requires a change to the mapping in which case the database manager determines from the database directory 1 if the page needs to be (re)mapped.

Detailed Description Text (40):
Remapping a page is illustrated in FIG. 13. Beginning at step 1304, the database manager determines whether a page needs to be remapped. If so, the database manager continues to step 1306 to prepare and issue a remapping request. Continuing to step 1308, a new data page is allocated. Continuing to step 1310, the directory 1 is updated and a new mapping is done. This results in a mapping change, not a change in the virtual address of a page.

Detailed Description Text (41):
An example of mapping on demand, as provided by one aspect of the present invention, is illustrated in FIG. 14 and is described using the IBM SQL/DS database management system operating on an IBM System/390 computer with the IBM VM/ESA operating system. Before updating a page, the IBM SQL/DS management system begins at step 1404 and copies the requested page to a buffer. It then reissues the MAPMDISK DEFINE request with the PAGEVIEW=ZERO parameter before copying the data back from the local buffer to the data space 608a-608q. Note, PAGEVIEW=ZERO informs the VM/ESA operating system that it should provide an empty real storage page for that data space 608a-608q page rather than reading the contents of the disk 2 when the page is first referenced.

Detailed Description Text (50):
In the prior art, mechanisms exist to request that changes to a database be written to disk 2 while the database manager is waiting. However, such mechanisms cause significant delays at a checkpoint time because many pages may be needed to be written to disk 2. To resolve this problem, some operating systems have implemented a mechanism whereby the database manager specifies a subset of the database to be made current on disk 2 (i.e. to be written to disk 2 if the pages in memory were modified). Further, the mechanism is said to be asynchronous in that a save request is issued, and processing continued, while the I/O operations are performed. An indicator is presented to the database manager when all I/O operations needed for that SAVE have completed.

Detailed Description Text (51):
One embodiment of the present invention takes advantage of the above mechanism to implement a partial save. Like mapping on demand which allows the database manager to delay the mapping to the first reference of a page, the invention provides for a partial save which allows the database manager to request page saves at short intervals and for only the modified pages of a data object.

Detailed Description Text (52):
Partial save is illustrated in FIGS. 15-17 and can be described using the IBM SQL/DS database management system as follows:

Detailed Description Text (54):
In step 1704, when the counter reaches a predefined limit (set by the user) the database manager proceeds to step 1706 and issues a MAPMDISK SAVE using as the list of addresses to save, the list corresponding to the pages represented by the "ON" bits in the section.sub.-- is.sub.-- modified bit map 1502. Step 1706 is described

in more detail below. The database manager then continues to step 1708 and resets the counter to 0. Since the request is asynchronous, the database manager continues processing while the I/O operations take place. The value of the counter determines the frequency with which the partial saves are issued (a lower counter causes more frequent partial saves). The counter also determines the maximum number of outstanding page sets for which the SAVE has been issued to save all the database pages on disk 2.

Detailed Description Text (62):
For each SAVE request issued, the database manager keeps the control block which was built to issue the request until the request completes. On completion, if the operating system indicates that some of the I/O operations were not completed, due to a hardware problem, the database manager takes corrective action.

Detailed Description Text (64):
With this invention, the database manager I/O function is bypassed. Furthermore, page I/O operations are implemented so that the operating system informs the database manager that a paging I/O operation is needed (and informs it when it is completed), thereby allowing the database manager to perform other tasks in parallel with the I/O operation. This technique results in very significant improvements in database response times without compromising data integrity.

Detailed Description Paragraph Table (3):
TABLE 3 _____ If Data.sub.-- space.sub.-- pointer (Data.sub.-- space.sub.-- number) = 0 (this data space does not already exist) then do; Create the data space Set Data.sub.-- space.sub.-- pointer(Data.sub.-- space.sub.-- number) to the sub-data space pointer value used to address that data space. Allocate the segment valid bit map for that data space and initialize it to off Allocate the section.sub.-- is.sub.-- modified bit map for that data space of size (m/n) and initialize it to off. Set Modified.sub.-- group.sub.-- count (Data.sub.-- space.sub.-- number) to 0. end; Similarly the page modification procedure can be represented in pseudocode as shown as follows: Modify the page. If Modified.sub.-- group.sub.-- count(Data.sub.-- space.sub.-- number) > Save.sub.-- interval and Section.sub.-- is.sub.-- modified (page-address/(n*y)) is on then do; Initialize a save request Do i = 0 to (x/n) - 1 If section.sub.-- is.sub.-- modified(i) is on then Add pages from (i*n) to ((i*n)+n) to the save request end; Reset Section.sub.-- is.sub.-- modified to off Issue the save request without waiting for completion end; If Section.sub.-- is.sub.-- modified(page-address/(n*y) is off then do; /* none of the pages in this section were modified */ Increment modified.sub.-- group.sub.-- count(Data.sub.-- space.sub.-- number) by Set Section.sub.-- is.sub.-- modified(page-address/(n*y) on. end; Similarly the checkpoint procedure can be represented in pseudocode as shown as follows: Do j = 1 to max(Data.sub.-- space.sub.-- number) Initialize a save request Do i = 0 to (x/n) - 1 If section.sub.-- is.sub.-- modified(i) for data space(j) is on Add pages from (i*n) to ((i*n)+n) to the save request End; Reset Section.sub.-- is.sub.-- modified for data space(j) to off Issue the save request without wait for completion End; Wait for all outstanding save requests to complete.

Current US Cross Reference Classification (1):
707/200

CLAIMS:

1. A method for modifying a page of a data object contained in a database and stored in one or more database storage disks, wherein the database comprises one or more database data objects, wherein the database is accessed via a contiguous data space representation, the contiguous data space being represented by a plurality of concatenated sub-data spaces, wherein each sub-data space is a virtual data space of a maximum size that is addressable by a computer operating system, each sub-data

space comprising a plurality of data segments, each data segment comprising a plurality of pages, wherein the pages in the contiguous data space representation are contiguous, each page comprising a known number of addressable storage locations, wherein the contiguous data space, the sub-data spaces, the data segments, and the pages are addressable by a database management system, wherein the address of a page to be modified has been determined to be placed in a sub-data space, the method comprising the steps of:

(1) mapping a data segment of the data object from a database storage disk to a data segment of the sub-data space, wherein said data segment from said database storage disk contains the page to be modified;

(2) creating a modified section bit map for the sub-data space, wherein said modified section bit map comprises a plurality of bits, each bit representing a group of pages of the sub-data space and indicating whether one or more pages in said group has been modified;

(3) indicating that the page to be modified has been modified, thereby creating a modified page, by setting a bit in said modified section bit map to a predetermined value, wherein said bit represents a group containing said modified page;

(4) generating a modified section count representing a number of bits in said modified section bit map that are equal to said predetermined value; and

(5) issuing an asynchronous request to save one or more groups of pages of the sub-data space corresponding to bits in said modified section bit map that are equal to said predetermined value when said modified section count corresponds to a modified section threshold.

2. A method for modifying a page of a data object contained in a database and stored in one or more database storage disks, wherein the database comprises one or more database data objects, wherein the database is accessed via a contiguous data space representation, the contiguous data space being represented by a plurality of concatenated sub-data spaces, wherein each sub-data space is a virtual data space of a maximum size that is addressable by a computer operating system, each sub-data space comprising a plurality of data segments, each data segment comprising a plurality of pages, wherein the pages in the contiguous data space representation are contiguous, each page comprising a known number of addressable storage locations, wherein the contiguous data space, the sub-data spaces, the data segments, and the pages are addressable by a database management system, wherein the address of a page to be modified has been determined to be placed in a sub-data space, the method comprising the steps of:

(1) mapping a data segment of the data object from a database storage disk to a data segment of the sub-data space, wherein said data segment from said database storage disk contains the page to be modified;

(2) creating a modified section bit map for the sub-data space, wherein said modified section bit map comprises a plurality of bits, each bit representing a group of pages of the sub-data space and indicating whether one or more pages in said group has been modified;

(3) indicating that the page to be modified has been modified, thereby creating a modified page, by setting a bit in said modified section bit map to a predetermined value, wherein said bit represents a group containing said modified page;

(4) generating a modified section count representing a number of bits in said modified section bit map that are equal to said predetermined value; and

(5) issuing an asynchronous request to save one or more groups of pages of the sub-

data space corresponding to bits in said modified section bit map that are equal to said predetermined value when said modified section count corresponds to a modified section threshold;

wherein, if the page to be modified requires a new mapping to said database storage disk, the method comprising the further steps of:

(6) copying the page to be modified to another work space in computer memory and modifying the page to be modified in said another work space in computer memory;

(7) allocating a new database storage page in said database storage disk;

(8) determining whether a mapping request to map the page to be modified to said new database storage page in said database storage disk can be combined with a prior mapping request, wherein said prior mapping request maps a second page in the sub-data space to a second database storage page in said database storage disk and the page to be modified is adjacent to said second page in the sub-data space;

(9) issuing said prior mapping request if step (8) determines that said mapping request cannot be combined with said prior mapping request and said prior mapping request exists;

(10) copying said second page of said prior mapping request from said another work space in computer memory if step (9) issues said prior mapping request;

(11) preparing said mapping request if step (9) issues said prior mapping request or said prior mapping request does not exist; and

(12) combining said mapping request with said prior mapping request if step (8) determines that said mapping request can be combined with said prior mapping request.

3. The method of claim 1, further comprising the step of:

(6) issuing a synchronous request to save one or more groups of pages of each said sub-data space corresponding to bits in each said modified section bit map that are equal to said predetermined value when the database management system determines that a checkpoint is required, wherein the database management system waits for all outstanding asynchronous requests to complete.

4. A computer system for modifying a page of a data object contained in a database and stored in one or more database storage disks, wherein the database comprises one or more database data objects, wherein the database is accessed via a contiguous data space representation, the contiguous data space being represented by a plurality of concatenated sub-data spaces, wherein each sub-data space is a virtual data space of a maximum size that is addressable by a computer operating system, each sub-data space comprising a plurality of data segments, each data segment comprising a plurality of pages, wherein the pages in the contiguous data space representation are contiguous, each page comprising a known number of addressable storage locations, wherein the contiguous data space, the sub-data spaces, the data segments, and the pages are addressable by a database management system, wherein the address of a page to be modified has been determined to be placed in a sub-data space, comprising:

mapping means for mapping a data segment of the database data object from a database storage disk to a data segment of the sub-data space, wherein said data segment from said database storage disk contains the page to be modified;

bit map means for creating a modified section bit map for the sub-data space, wherein said modified section bit map comprises a plurality of bits, each bit

representing a group of pages of the sub-data space and indicating whether one or more pages in said group has been modified;

indicating means for indicating that the page to be modified has been modified, thereby creating a modified page, by setting a bit in said modified section bit map to a predetermined value, wherein said bit represents a group containing said modified page;

generating means for generating a modified section count representing the number of bits in said modified section bit map that are equal to said predetermined value; and

issuing means for issuing an asynchronous request to save one or more groups of pages of the sub-data space corresponding to bits in said modified section bit map that are equal to said predetermined value when said modified section count corresponds to a modified section threshold.

6. The computer system according to claim 4, further comprising:

second issuing means for issuing a synchronous request to save one or more groups of pages of each said sub-data space corresponding to bits in each said modified section bit map that are equal to said predetermined value when the database management system determines that a checkpoint is required, wherein the database management system waits for all outstanding asynchronous requests to complete.